PAPER Special Section on Cryptography and Information Security

Efficient Algorithms for Sign Detection in RNS Using Approximate Reciprocals

Shinichi KAWAMURA[†,††,†††a)], ***Fellow***, **Yuichi KOMANO**[††], **Hideo SHIMIZU**[††], ***Members***, **Saki OSUKA**[††††], ***Nonmember***, **Daisuke FUJIMOTO**[††††], **Yuichi HAYASHI**[†††,††††], ***Members***, ***and*** **Kentaro IMAFUKU**[†††], ***Nonmember***

SUMMARY The residue number system (RNS) is a method for representing an integer x as an n-tuple of its residues with respect to a given set of moduli. In RNS, addition, subtraction, and multiplication can be carried out by independent operations with respect to each modulus. Therefore, an n-fold speedup can be achieved by parallel processing. The main disadvantage of RNS is that we cannot efficiently compare the magnitude of two integers or determine the sign of an integer. Two general methods of comparison are to transform a number in RNS to a mixed-radix system or to a radix representation using the Chinese remainder theorem (CRT). We used the CRT to derive an equation approximating a value of x relative to M, the product of moduli. Then, we propose two algorithms that efficiently evaluate the equation and output a sign bit. The expected number of steps of these algorithms is of order n. The algorithms use a lookup table that is $(n+3)$ times as large as M, which is reasonably small for most applications including cryptography.

key words: *Chinese remainder, residue number system, sign detection, comparison*

1. Introduction

RNS (residue number system) is a method for representing an integer x as an n-tuple of its residues with respect to a given set of bases $\{m_1, m_2, \cdots, m_n\}$. The main feature of RNS is that addition, subtraction, and multiplication can be carried out by independent addition, subtraction, and multiplication with respect to each base element, which enables fast computation via parallel processing. Due to this property, a lot of studies have been conducted to implement computation on integers with hundreds to thousands of bits, a size necessary for public-key cryptography, since around 2000 [1], [2]. One reason for this timing is that an efficient base extension algorithm was proposed in [3], which is useful for implementing Montgomery multiplication in RNS. However, how to efficiently compare magnitude of two integers or determine the sign of an integer in RNS is still an unsolved problem. Namely, comparison in RNS requires more computation steps than other operations such as multiplication. It is not unusual to avoid comparison in RNS. For example, Bigou et al. studied the extended Euclidean algorithms, which require no comparison [4], [5].

General methods of comparison are classified into methods that either transform a number in RNS to a mixed-radix system (MRS) or to a radix representation using the Chinese remainder theorem. Garner proposed a method to transform RNS to MRS in $O(n^2)$ steps [6], which is regarded as the most efficient way among general methods. In fact, Knuth conjectured in his book that "there is little hope of finding a substantially better method [than Garner's], since the range of a modular number depends essentially on all bits of all the residues" ([7], p.291. Words within [] were added by the present authors).

Vu proposed a sign-detection method that is more efficient than Garner's in some restricted cases [8]. His idea is to evaluate x/M instead of x, using the Chinese remainder theorem, where $M = m_1 m_2 \cdots m_n$. Recently, a new sign-detection algorithm was proposed based on the Chinese remainder theorem [9], which is superior to any methods proposed before [9]. Both methods in [8] and [9] use lookup tables, which become enormous unless the bit size of the base elements is sufficiently small. Memory size for [9] is evaluated as $O((\log_2 M)^3/(\log_2 \log_2 M)^2)$ in bits.

We propose a new sign-detection algorithm based on the Chinese remainder theorem. In our algorithm, we evaluate $1/m_i$ by approximation and obtain a computational complexity of $O(n)$ with memory complexity, $O\left((\log_2 M)^2\right)$. We apply two approximation methods: one is based on a power series, and the other is based on a finite-length reciprocal table. In our algorithms, the approximation error is made smaller step-by-step until we can determine a sign bit. Once a sign is determined, the algorithm halts. Therefore, the number of computation steps can be limited. To make the algorithm implementation-friendly, we design it to be word-oriented.

A lot of research effort has considered specific sets of moduli such as $\{2^w+1, 2^w, 2^w-1\}$ [10]–[13]. However, these moduli sets have relatively small size and do not necessarily fit to the optimization for fast cryptographic implementation such as [14]–[16]. Therefore, we focus on moduli sets that are scalable in size and easy to use for applications such as cryptography. The problem we try to solve is to propose a sign-detection algorithm that works for general bases with more efficiency and less memory than conventional algorithms.

Manuscript received March 16, 2020.
Manuscript revised June 12, 2020.

[†]The author is with ECSEC Technical Research Association, Tokyo, 101-0054 Japan.
[††]The authors are with Toshiba Corporation, Kawasaki-shi, 212-8582 Japan.
[†††]The authors are with National Institute of Advanced Industrial Science and Technology, Tokyo, 135-0064 Japan.
[††††]The authors are with Nara Institute of Science and Technology, Ikoma-shi, 630-0192 Japan.
a) E-mail: shinichi.kawamura@aist.go.jp

DOI: 10.1587/transfun.2020CIP0020

In Sect. 2, we describe the notation and background of our research. Section 3 explains the principle of our method. We propose a method based on a power series in Section 4, and a method based on a reciprocal table in Sect. 5. Section 6 evaluates the computational complexity and memory size. Section 7 concludes the paper.

2. Notation and Background

2.1 Notation

The following notation is used in this paper.
w: the bit size of a word in a given computer.
$\langle x\rangle_m = x \bmod m$, where $\langle x\rangle_m \in [0, m)$.
Base $B = \{m_1, \ldots, m_n\}$, where $\gcd(m_i, m_j) = 1 (i \neq j)$.
Here, we assume

$$m_i = 2^w - \mu_i.$$

μ_i is an integer in the range, $0 \leq \mu_i < 2^{\lfloor w/2 \rfloor}$.

$$M = \prod_{i=1}^{n} m_i$$
$$M_i = M/m_i$$

$\left\langle x^{-1}\right\rangle_{m_i}$: a multiplicative inverse of x modulo m_i, if $\gcd(m_i, x) = 1$ holds.
$\{x\}_B = [\langle x\rangle_{m_1}, \ldots, \langle x\rangle_{m_n}]$: RNS representation of x.
Transpose T:

$$\{x\}_B^T = \begin{bmatrix} \langle x\rangle_{m_1} \\ \vdots \\ \langle x\rangle_{m_n} \end{bmatrix}$$

$\langle x\rangle_m \otimes \langle y\rangle_m \triangleq \langle xy\rangle_m$
$\{x\}_B \otimes \{y\}_B \triangleq \{xy\}_B = [\langle xy\rangle_{m_1}, \ldots, \langle xy\rangle_{m_n}]$
$\{x\}_B + \{y\}_B \triangleq \{x + y\}_B = [\langle x + y\rangle_{m_1}, \ldots, \langle x + y\rangle_{m_n}]$
A constant W associated with base B:

$$W = \left\langle \sum_{i=1}^{n} \left\langle M_i^{-2}\right\rangle_{m_i} M_i \right\rangle_M$$

$\{W\}_B = \left[\left\langle M_1^{-1}\right\rangle_{m_1}, \ldots, \left\langle M_n^{-1}\right\rangle_{m_n}\right]$: The RNS representation of W.
RNS(x): RNS representation of x, not specifying a base.
MRS(x): MRS representation of x, not specifying a base.
$\langle x\rangle_1 = x - \lfloor x\rfloor$: A fractional part of a real number x. This is a natural extension of the symbol $\langle x\rangle_m = x \bmod m = x - m\lfloor x/m\rfloor$, defined for integers x, $m > 1$, since the right-hand sides coincide with each other if $m = 1$ is chosen.

$$x = \lfloor x\rfloor + \langle x\rangle_1$$

holds since x is a sum of an integer part $\lfloor x\rfloor$ and a fractional part $\langle x\rangle_1$.

For an arbitrary integer a, the following equation holds:

$$\frac{\langle a\rangle_m}{m} = \left\langle \frac{a}{m}\right\rangle_1.$$

Recall the relation below for proof.

$$a = m\left\lfloor \frac{a}{m}\right\rfloor + \langle a\rangle_m$$

The equation is apparent since $\lfloor a/m\rfloor$ and $\langle a\rangle_m$ represent a quotient and a residue of a/m, respectively.

The dot product (or inner product) may be used for the algorithm description.

$$\begin{bmatrix} a_1 & a_2 & \ldots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} a_i x_i$$

k-bit right shift of an integer x:

$$x \gg k = \left\lfloor \frac{x}{2^k}\right\rfloor.$$

2.2 Sign Detection Based on MRS

We first define a sign function and a number with reverse sign.
Definition 1:

Let $\{x\}_B$ be an RNS representation of an integer $x \in [0, M-1]$. A sign function of $\{x\}_B$ is defined by

$$\text{sign}\,(\{x\}_B) = \begin{cases} 0, & \text{if } x < M/2 \\ 1, & \text{if } x \geq M/2 \end{cases}$$

■

Definition 2:

Let $\{a\}_B$ be an RNS representation of an integer $a \in [0, M-1]$. A number with the reverse sign of $\{a\}_B$ is given by

$$\{-a\}_B = \{M - a\}_B.$$

■

When Definition 1 is to be implemented, the problem is that the if-clause cannot be evaluated efficiently for a given $\{x\}_B$.

Let $[y_1, y_2 \cdots y_n]$ be an MRS representation of a number represented as $[x_1, x_2 \cdots x_n]$ in RNS. Then the following equation holds:

$$x = y_n m_{n-1} \cdots m_1 + \cdots + y_2 m_1 + y_1.$$

In MRS, comparison of two integers can be carried out by comparing each digit y_i from the most significant digit to the least significant one, as with ordinary radix representation. Therefore, we can compute a sign function as follows.

1. Convert RNS(x) to MRS(x) using Garner's algorithm [4].
2. Compare MRS(x) with a precomputed value, MRS($\lceil M/2\rceil$).

$$\text{sign}\,(\{x\}_B) = \begin{cases} 0, & \text{if MRS } (x) < \text{MRS}(\lceil M/2\rceil) \\ 1, & \text{if MRS } (x) \geq \text{MRS}(\lceil M/2\rceil) \end{cases}$$

The amount of computation necessary in each of the above steps is evaluated as the following.
Step 1. Modular multiplication: $n(n-1)/2$ times,
Step 2. Comparison of w-bit words: Minimum once, maximum n times.

The most time-consuming part of this algorithm is modular multiplications, the number of which is estimated as $O(n^2)$. Note that in step 1, the first word computed is the least significant one, and the last word is the most significant one. Consequently, we cannot proceed to step 2 before computing all the words of MRS. As far as sticking to the basic operations permitted in RNS, there is no known comparison algorithm with complexity lower than $O(n^2)$.

2.3 Sign Function and x/M

We can modify Definition 1 to derive Definition 3, in which equations in the if-clauses are divided by M on both sides, without affecting the sign function.
Definition 3:

$$\text{sign}\,(\{x\}_B) = \begin{cases} 0, & \text{if } x/M < 1/2 \\ 1, & \text{if } x/M \geq 1/2 \end{cases}$$

■

According to Definition 3, we can efficiently compute the sign function if x/M is evaluated efficiently from $\{x\}_B$. Let the binary representation of x/M be defined by

$$\frac{x}{M} = \sum_{i=1}^{\infty} b_{-i} \cdot 2^{-i},$$

where $b_{-i} \in \{0, 1\}$. Usually, the following relation holds.

$$b_{-1} = 0 \Longleftrightarrow x/M < 1/2$$
$$b_{-1} = 1 \Longleftrightarrow x/M \geq 1/2$$

In this case, sign $(\{x\}_B)$ is given by the first bit after the decimal point of x/M.

An exception can occur when $x = M/2$. If x/M is represented as $1 \cdot 2^{-1}$, then the rule above succeeds. But 1/2 can also be represented by a repeating decimal as

$$\frac{1}{2} = \sum_{i=2}^{\infty} 1 \cdot 2^{-i}.$$

In such a case, $b_{-1} = 0$, and the above decision rule fails.

The Chinese remainder theorem is known as a way to compute a radix representation of x from an RNS representation. The equation below is an expression of the Chinese remainder theorem

$$x = \left\langle \sum_{i=1}^{n} \left\langle x_i M_i^{-1} \right\rangle_{m_i} M_i \right\rangle_M, \tag{1}$$

where $x_i = \langle x \rangle_{m_i}$. If we divide both sides by M and replace a variable with $\xi_i = \left\langle x_i M_i^{-1} \right\rangle_{m_i}$, we obtain

$$\frac{x}{M} = \left\langle \sum_{i=1}^{n} \frac{\xi_i}{m_i} \right\rangle_1. \tag{2}$$

The right-hand side means an operation to provide the fractional part of the number in the parenthesis, or to truncate the integer part. By using $\{W\}_B$, defined in Sect. 2.1, we can express ξ_i simply as

$$\begin{bmatrix} \xi_1 & \xi_2 & \cdots & \xi_n \end{bmatrix} = \{x\}_B \otimes \{W\}_B.$$

We can construct a sign-detection algorithm by applying (2) to the if-clause of Definition 3. This idea was first proposed in [8]. Although the equation evaluated in [8] is slightly different from (2) in that both sides are doubled (Equation (10) of [8]), its principle is the same as Theorem 1, which will appear in Sect. 3. In [8], the equation is evaluated using a lookup table, which has an entry ξ_i/m_i with a precision of $\log_2 nM$ bits, addressed by x_i. Since the address is x_i, the table entries become larger as the bit size of the base elements increases. Consequently, the application is limited to those with small moduli.

Recently, a new sign-detection algorithm with complexity $O(n)$ was proposed in [9]. The algorithm uses a lookup table with very coarse precision. Nevertheless, it shares the same problem with the algorithm in [8], since x_i is used as the address to a table entry. It is recommended in [9] that the base elements should be consecutive prime numbers starting from 3 to make the elements as small as possible.

For cryptographic applications, it is often the case that the bit length of each base element is chosen to be close to the word length w of a given computer to make each operation efficient. No sign-detection algorithm with a moderate table size has been proposed, even in such cases as $w \geq 32$. Equation (2) is also used in [17], but procedure of sign detection is less efficient.

3. Principle of Proposal

3.1 Approximation Function

Let $G(x)$ denote the value in the parenthesis of (2). Then,

$$G(x) = \sum_{i=1}^{n} \frac{\xi_i}{m_i}.$$

In addition, the approximation of G is denoted by $G(x, d)$, where the second argument is a non-negative integer that represents the degree of approximation. We define the approximation error by

$$e(x, d) = G(x) - G(x, d).$$

We choose an approximation function $G(x, d)$ such that the following three conditions are satisfied.

(i) $e(x, d) \geq e(x, d+1) \geq 0$
(ii) $\lim_{d \to \infty} e(x, d) = 0$
(iii) $e(0, d) = 0$

This requires that $e(x, d)$ decreases monotonically as d increases, that $e(x, d) = 0$ with d infinite, and that the error is 0 when $x = 0$.

Let $e(d)$ be defined by

$$e(d) = \max_x e(x, d).$$

Then we obtain the following equation.

$$0 \le e(x, d) \le e(d)$$

Conditions (i) and (ii) ensure that we can make the error as close to 0 as we like. Now we get Lemma 1 and Theorem 1.

Lemma 1:

If an approximation function and its error satisfy conditions (i) and (ii), there exists an integer δ such that

$$0 \le e(d) \le \varepsilon$$

holds for an arbitrary small real number $\varepsilon > 0$ and all integers d satisfying $d \ge \delta$. ■

Theorem 1:

If the approximation error $e(x, d)$ satisfies Conditions (i)–(iii), there exists an integer δ satisfying

$$e(\delta) \le \frac{1}{2M}.$$

Then, for any $x \in [0, M-1]$ excluding $x = M/2$, the first bit after the decimal point of

$$\langle G(x, \delta)\rangle_1$$

is identical to that of $\langle G(x)\rangle_1$, which equals the sign bit defined in Definitions 1 and 3. ■

Refer to Appendix A for the proof. We exclude the case $x = M/2$ since there is a possibility that we cannot determine the sign by b_{-1} alone due to the occurrence of a repeating decimal. We will discuss how to deal with the case $x = M/2$ in Sect. 4.

3.2 Approximation Error and Sign Detection

We explain the mechanism to determine the sign of x from $\langle G(x, d)\rangle_1$. Figure 1 is a diagram showing the value of $\langle G(x, d)\rangle_1$ computed for a given x. Points $\mathrm{P}(x) = (x, \langle G(x, d)\rangle_1)$ appear in the area between the lines $y = (1/M)x$ and $y = (1/M)x - e$. In addition, they also appear in the triangular area above the line $y = (1/M)x + (1 - e)$. Here, the symbol e is a simplified expression of the error bound $e(d)$. To determine a sign, we first compute $\langle G(x, d)\rangle_1$ from $\{x\}_B$, then evaluate the value according to the following rules:

$$\begin{aligned}
&\langle G(x, d)\rangle_1 \in [0, 1/2 - e) \Rightarrow \mathrm{sign}(x) = 0,\\
&\langle G(x, d)\rangle_1 \in [1/2, 1 - e) \Rightarrow \mathrm{sign}(x) = 1,\\
&\langle G(x, d)\rangle_1 \in [1/2 - e, 1/2) \cup [1 - e, 1)\\
&\qquad\qquad \Rightarrow \text{indeterminate}.
\end{aligned}$$

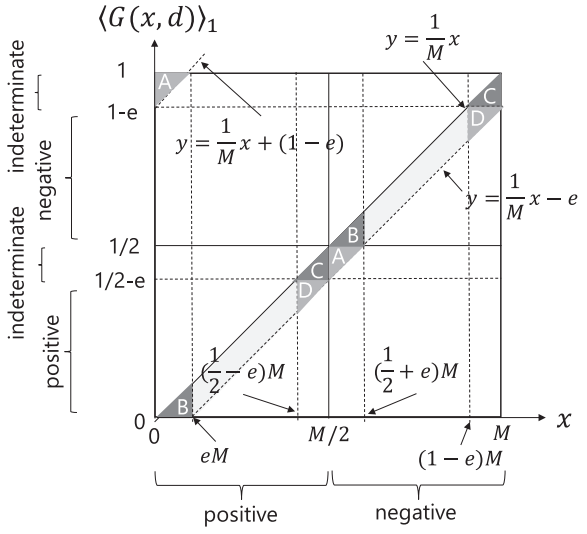


Fig. 1 Diagram of x versus approximate x/M.

If $\langle G(x, d)\rangle_1 \in [1/2 - e, 1/2)$ occurs from the indeterminate case, we can narrow the range of x as $((1/2) - e)M \le x \le ((1/2) + e)M$. But, if $\mathrm{P}(x)$ is in the area C, the correct sign of x is 0. If, instead, $\mathrm{P}(x)$ is in the area A, the correct sign of x is 1. Thus, we cannot tell the correct sign. Similarly, if $\langle G(x, d)\rangle_1 \in [1 - e, 1)$ occurs, we can only conclude that $x \in [0, eM]$ or $x \in [(1 - e)M, M)$, and cannot determine the sign. We can rephrase the condition of Theorem 1, $e(\delta) \le 1/2M$, as the condition that no point appears in the area A or in area C in Fig. 1. The analysis here also tells us that relatively high precision, or a larger d, is necessary for sign detection near $x = M/2$, $x = 0$, and $x = M$. Less precision, or smaller d, is required for x away from them.

We will propose two candidates for the approximation function $G(x, d)$; one is based on a power series and the other is based on a reciprocal table.

4. Method Based on Power Series

To derive a concrete algorithm, the following three points should be considered.

- Choice of an approximation function that satisfies the condition in Theorem 1.
- Proposal of efficient algorithm to compute the approximation function.
- Moderate memory size.

4.1 Choice of Approximation Function

We pose a condition that is frequently used in the cryptographic implementation on the base elements, specifically,

$$m_i = 2^w - \mu_i,$$

where μ_i is an integer in the range $0 \le \mu_i < 2^{\lfloor w/2 \rfloor}$ and w is the bit size of a word for a given computer. With such a modulus, modular operations can be easily implemented.

Further optimization regarding μ_i may be possible for efficient implementation (e.g. [15]). The proposed algorithm can be combined with such optimization, if necessary.

The reciprocal of m_i can be expanded as a power series

$$\frac{1}{m_i} = \frac{1}{2^w} \cdot \frac{1}{1-\mu_i/2^w} = \frac{1}{2^w} \sum_{k=0}^{\infty} \left(\frac{\mu_i}{2^w}\right)^k.$$

Note that the equation holds for any case, including $\mu_i = 0$, if we define $0^0 = 1$.

Substituting an infinite power series for the equation of $G(x)$ and truncating the tail after the d-th power, we obtain the approximation function for $G(x)$ as

$$G(x,d) = \sum_{i=1}^{n} \frac{\xi_i}{2^w} \sum_{k=0}^{d} \left(\frac{\mu_i}{2^w}\right)^k. \tag{3}$$

The approximation error $e(x,d)$ is given by

$$e(x,d) = \sum_{i=1}^{n} \frac{\xi_i}{m_i} \left(\frac{\mu_i}{2^w}\right)^{d+1}.$$

If x varies, ξ_i moves in the range $0 \le \xi_i \le m_i - 1$. Therefore, $e(x,d)$ ranges in

$$0 \le e(x,d) \le e(d) = \sum_{i=1}^{n} \left(1 - \frac{1}{m_i}\right) \left(\frac{\mu_i}{2^w}\right)^{d+1}. \tag{4}$$

As d increases, the upper bound decreases exponentially. This approximation fulfills the conditions (i)–(iii) in Sect. 3.

To simplify the expression of $G(x,d)$, we define g as

$$g(x,k) = \sum_{i=1}^{n} \xi_i \cdot \mu_i^k. \tag{5}$$

Then, $G(x,d)$ is rewritten as

$$G(x,d) = \sum_{k=0}^{d} \frac{1}{2^{w(k+1)}} g(x,k). \tag{6}$$

Next, we find a parameter d that is large enough for sign detection. Since $M \approx 2^{nw}$, the following equation holds:

$$e(n-1) = \sum_{i=1}^{n} \left(1 - \frac{1}{m_i}\right) \left(\frac{\mu_i}{2^w}\right)^n > \frac{1}{2^{nw}} > \frac{1}{2M}.$$

This means that $d = n - 1$ is not sufficiently large. If $d = n$, then

$$e(n) = \frac{1}{2^{(n+1)w}} \sum_{i=1}^{n} \left(1 - \frac{1}{m_i}\right) \mu_i^{n+1}.$$

This implies there is a possibility that $e(n)$ will satisfy the condition in Theorem 1 if μ_i is small to some extent. In other words, $d \ge n$ is a necessary condition for the assumption of Theorem 1 to be fulfilled. Let us assume that we can find μ_i and w that satisfy the assumption of Theorem 1 for $d = n$. Even if this is not true, we can make the error sufficiently small if we choose a bigger d. In this case, however, the number of computation steps increases as well. In addition, the term μ_i^k in (5) becomes larger and there is the possibility that an operation larger than w bits is necessary.

Let us estimate the number of steps necessary to compute (3) for $d = n$. The number of terms that appear in a power series is n for each base. n multiplications are necessary to multiply n terms by ξ_i. Since the suffix i has n different values, we have $O\left(n^2\right)$ multiplications in total. This is not better than that of MRS. To improve the computational complexity, we must devise an efficient algorithm to evaluate the approximate equation.

4.2 SDPS Algorithm

We propose a sign-detection algorithm based on Theorem 1 and name it SDPS (sign detection using a power series), the pseudocode of which is shown in Fig. 2. The variable gx_k corresponds to a function $g(x,k)$ defined by (5). To compute $G(x,d)$ efficiently, SDPS is designed according to the following policies.

- Compute μ_i^k in advance as a lookup table for $1/m_i$.
- Start computing from the most significant word, which includes a sign bit.
- Stop computing as soon as a sign bit is determined.

These make it possible for the average number of computation steps to be $O(n)$. To realize this, we must establish a way to decide whether the sign bit has been determined.

In Fig. 2, the algorithm stops either at step 13, in the middle of the for-loop, or at step 17, after the n-th loop is finished. At step 11, the if-clause is executed if *carry* equals 1 or *sum* is not all 1s. In the case when *carry* equals 1, *carry* travels to the position of the sign bit, b_{-1}, and the sign bit is fixed ever after. In the case when *sum* is not all 1s, *sum* will not produce a new carry since a carry from the less significant block will stop within *sum*. Hence, the sign bit is fixed in this case as well. The basis of this decision rule is the following simple property of carry propagation.

Let B, C be two binary numbers whose i-th bit is represented by $b_i, c_i \in \{0,1\}$, respectively. The suffix is an integer and a larger suffix represents a more significant bit. Suppose b_i is the bit of interest and $i > j$. If a carry c_j is added to the j-th bit of B, then b_i changes if and only if $c_j = 1$ and all bits from b_{i-1} to b_j are 1.

The bit alignment of values computed in SDPS is presented in Fig. 3, where the left end is the decimal point and lower bits are located in the right-hand direction. In Fig. 3, *low*(0) and *high*(1) are added first, and *sum*(1) and *carry*(1) are obtained. This procedure is continued to less significant blocks until the sign bit is fixed. In SDPS, the bit size of *high*(k) becomes bigger as the parameter k increases due to the bit size of μ_i^k. If *high*(k) becomes too big, SDPS will not give the correct sign. To circumvent such cases, we impose the following two conditions at step 7.

1. *high*(1) $< 2^{w-1}$ holds in the first loop

Sign($\{x\}_B, B$)
Required: sign = 1, if $\frac{x}{M} > \frac{1}{2}$ and sign = 0, if $\frac{x}{M} < \frac{1}{2}$.
0:
$$\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix} \leftarrow (\{x\}_B \otimes \{W\}_B)^T$$
1: // Ordinary inner product
$$gx\_k \leftarrow [1\ 1\ \ \ldots\ \ 1]\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$$
2: $low \leftarrow \langle gx\_k \rangle_{2^w}$
3: $sign \leftarrow low \gg (w-1)$
4: if $sign = 0$, then $low \leftarrow low + 2^{w-1}$ end if
5: for k = 1,n do
6: // Ordinary inner product
$$gx\_k \leftarrow [\mu_1^k\ \ \mu_2^k \ldots\ \ \mu_n^k]\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$$
7: $high \leftarrow gx\_k \gg w$
8: $z \leftarrow high + low$
9: $sum \leftarrow \langle z \rangle_{2^w}$
10: $carry \leftarrow z \gg w$
11: if $(carry = 1)$ OR $(sum \neq 2^w - 1)$ then
12: $sign \leftarrow sign$ XOR $carry$
13: return($sign$)
14: end if
15: $low \leftarrow \langle gx\_k \rangle_{2^w}$
16: end for
17: return($sign$)

Fig. 2 RNS sign-detection algorithm SDPS.

2. $high(k) < 2^w$ holds in the later loops

Condition 1 ensures that the first bit of $high(1)$ after the decimal point is 0, which means $high(1)$ does not directly modify the sign bit. Condition 2 ensures that the carry at step 10 is at most 1. It is possible to prove that these conditions are satisfied if

$$e(n) \leq \frac{1}{2M}$$

holds (Appendix B).

Finally, we assert by Theorem 2 that the SDPS algorithm outputs the same result as computed by $\langle G(x,n) \rangle_1$.

Theorem 2:

If

$$e(n) = \sum_{i=1}^{n}\left(1 - \frac{1}{m_i}\right)\left(\frac{\mu_i}{2^w}\right)^{n+1} \leq \frac{1}{2M}$$

holds, the return value of SDPS, except for the case $x = M/2$, is identical to the first bit after the decimal point of

$$\langle G(x,n) \rangle_1.$$

■

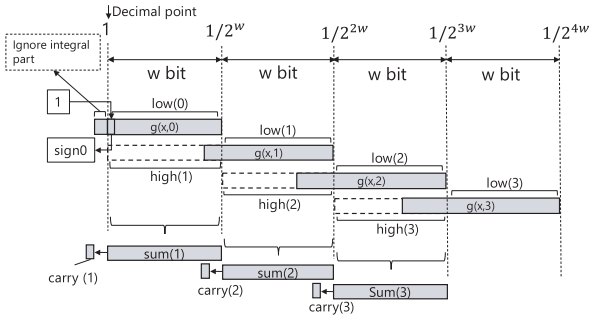


Fig. 3 Bit alignment of steps 1–10 of SDPS for $n = 3$.

In addition, we can derive two corollaries that have simpler assumptions.

Corollary 1:

Theorem 2 holds even if we replace the assumption by

$$\sum_{i=1}^{n} \mu_i^{n+1} < 2^{(w-1)}.$$

■

Corollary 2:

Theorem 2 holds even if we replace the assumption by

$$\max(\mu_i)^{n+1} \cdot n < 2^{(w-1)}.$$

■

Proof is that from the assumptions of Corollaries 1 and 2, we can derive the following equation:

$$e(n) < \frac{1}{2^{wn+1}} < \frac{1}{2M}.$$

The assumption of Corollary 1 can be violated for large n unless μ_i is adequately small. In Sect. 5, we will propose an algorithm with an easier constraint on μ_i so that the sign can be detected for a much wider range of bases.

4.3 Handling of $x = M/2$

SDPS excludes the case that $x = M/2$ for input. We consider how to deal with such a case.

1. Avoid $x = M/2$ by using odd M.
2. Use even M, and return 1 if the input is $x = M/2$. We give three ways to determine the sign.

   (a) Suppose m_1 is even. Then, $x = M/2$ can be represented as

   $$\{M/2\}_B = [m_1/2, 0, \ldots, 0].$$

   If this input is detected, return 1 immediately.

   (b) Choose $m_1 = 2^w$, that is, $\mu_1 = 0$, and run SDPS as usual. Then, SDPS returns 1.

   (c) Choose $m_1 = 2^w - \mu_1$, where μ_1 is a non-negative even number. In this case, SDPS computes 1/2 as a repeating decimal. We detect it by finding that the length of consecutive 1s is more than n words. This can be achieved by inserting the following code immediately

after step 14 of SDPS.
s1: if $k = n$ then return(1) end if

5. Method Based on Reciprocal Table

5.1 Choice of Approximation Function

We propose an algorithm that has fewer restrictions on the parameter μ_i than SDPS. We assume that the base has a form explained in Sect. 2.1.

A reciprocal table is produced by dividing the reciprocal of a modulus represented in binary into a sequence of words $h_i(k)$.

$$\frac{1}{m_i} = \sum_{k=1}^{\infty} h_i(k) \cdot 2^{-kw}$$
$$0 \le h_i(k) \le 2^w - 1$$

Substituting the table into (2) leads to

$$\frac{x}{M} = \left\langle \sum_{i=1}^{n} \left(\xi_i \sum_{k=1}^{\infty} h_i(k) \cdot 2^{-kw} \right) \right\rangle_1 .$$

To distinguish a new approximation function from $G(x, n)$, we use $H(x, d)$, which is defined as

$$H(x,d) = \sum_{k=1}^{d} h(x,k) \cdot 2^{-kw}$$
$$h(x,k) = \sum_{i=1}^{n} \xi_i \cdot h_i(k)$$
$$e_H(x,d) = \sum_{k=d+1}^{\infty} h(x,k) \cdot 2^{-kw}.$$

The upper bound of the error is estimated as

$$e_H(x,d) \le e_H(d) < n \cdot 2^{-(d-1)w}.$$

This means that we can make the error as close to 0 as we like by taking sufficiently large d. Thus, we can find an integer δ that satisfies assumption of Theorem 1. If we take $\delta = n + 2$, it follows that

$$e_H(n+2) < n \cdot 2^{-(n+1)w}.$$

In addition, if n and w satisfy the equation

$$n < 2^{w-1}, \tag{7}$$

we get

$$e_H(n+2) < 2^{w-1} 2^{-(n+1)w} = 2^{-nw-1} < \frac{1}{2M}.$$

This means that a sign bit is given by the first bit after the decimal point of $\langle H(x, n+2) \rangle_1$.

To efficiently evaluate H, we compute $h_i(k)$ in advance. The first two entries of $h_i(k)$ can be written as

$$h_i(1) = 1$$
$$h_i(2) = \mu_i \ \text{ or } \ \mu_i + 1$$

from analysis using a power series (Appendix C). We use these values to describe the algorithm in the next subsection. The value of $h_i(2)$ is described as $\bar{\mu}_i$, which equals μ_i or $\mu_i + 1$.

5.2 SDRT Algorithm

It takes $O(n^2)$ operations to compute a function $\langle H(x, n+2) \rangle_1$ with full accuracy. To reduce the order, we propose an algorithm similar to SDPS that controls precision adaptively and halts as soon as the sign bit is fixed. We name this SDRT (sign detection using reciprocal tables). Pseudocode and the bit alignment of SDRT are shown in Figs. 4 and 5, respectively. As discussed in the previous subsection, it is necessary to take a d equal to or larger than $(n+2)$ so that the error is sufficiently small. In SDRT, d is set to $(n + 3)$ to simplify the description of the algorithm.

In SDPS, the boundary of words coincides with that of processing; that is, the variable $sum(k)$ just fits between two word-boundaries in Fig. 3. In addition, it is a crucial point that the sign-detection at step 11 in Fig. 2 works correctly under the condition that the *carry* is at most 1. On the other hand, if the boundaries of words and processing were made to coincide in SDRT, then *carry* would become at most 2. In order to make *carry* at most 1, we shift the boundary of processing by 1 bit to the decimal point. This makes *carry* at most 1 and the sign detection works correctly in Fig. 4.

To confirm that *carry* is at most 1, we first consider (7) and derive the following equation:

$$(h_3 \gg 2w) < n < 2^{w-1}.$$

In addition, it is apparent that

$$\langle h_1 \rangle_{2^w} \le 2^w - 1$$
$$\langle h_2 \gg w \rangle_{2^w} \le 2^w - 1.$$

Then, we can estimate the maximum value of *tmp* at step 19 in Fig. 4 as less than $2^w + 2^{w-1} + 2^{w-2} - 1$. This proves that the *carry* is at most 1. As a result, we can use the same code for steps 21–24 in Fig. 4 as used in steps 11–14 in Fig. 2.

Let S be the approximate value of $\langle H(x, \infty) \rangle_1$ derived from SDRT. S is computed by the summation of carries and sums that are aligned as shown in Fig. 5. Since S is truncated at the $\{(n+1)w - 1\}$-th bit after the decimal point, its approximation error e_{RT} is

$$e_{RT} = \langle H(x, \infty) \rangle_1 - S < 2^{-(n+1)w+1} < \frac{1}{2M}.$$

Since e_{RT} satisfies the assumption of Theorem 1, we obtain the following Theorem.

Theorem 3:

If $n < 2^{w-1}$, then for any integer $x \in [0, M-1]$ except $x = M/2$, the return value of SDRT is identical to the first bit after the decimal point of

Sign($\{x\}_B, B$)
Required: sign = 1, if $\frac{x}{M} > \frac{1}{2}$ and sign = 0, if $\frac{x}{M} < \frac{1}{2}$
0:
$\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix} \leftarrow (\{x\}_B \otimes \{W\}_B)^T$
1: // Ordinary inner product
$h2 \leftarrow [1, \quad \dots \quad ,1]\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$
2:// Ordinary inner product
$h3 \leftarrow [\bar{\mu}_1, \quad \dots \quad ,\bar{\mu}_n]\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$
3: for k = 3, n+3 do
4: $h1 \leftarrow h2$
5: $h2 \leftarrow h3$
6: // Ordinary inner product
$h3 \leftarrow [h_1(k), \quad \dots \quad , h_n(k)]\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$
7: if k = 3 then
8: $body \leftarrow h1 + (h2 \gg w) + (h3 \gg 2w)$
9: $tail \leftarrow \langle body \rangle_2$
10: $sum \leftarrow \langle body \rangle_{2^w}$
11: $sign \leftarrow sum \gg (\mathrm{w} - 1)$
12: if $(\langle sum \rangle_{2^{w-1}} \gg 1) \neq 2^{w-2} - 1$ then
13: return($sign$)
14: end if
15: else
16 $body \leftarrow \langle h1 \rangle_{2^w} + \langle h2 \gg w \rangle_{2^w} + \langle h3 \gg 2w \rangle_{2^w}$
17: $tmp \leftarrow (body + tail * 2^w) \gg 1$
18: $tail \leftarrow \langle body \rangle_2$
19: $carry \leftarrow tmp \gg w$
20: $sum \leftarrow \langle tmp \rangle_{2^w}$
21: if $(carry = 1)$ OR $(sum \neq 2^w - 1)$ then
22: $sign \leftarrow sign$ XOR $carry$
23: return($sign$)
24: end if
25: end if
26: end for
27: return($sign$)

Fig. 4 RNS sign-detection algorithm SDRT.

$\langle H(x, \infty) \rangle_1$.

■

If $x = M/2$ occurs in SDRT, the same action as explained in Sect. 4.3 can be taken. As a special case, the action corresponding to 2-(c) is to insert the following code immediately after step 24 in Fig. 4.

s1: if $k = n + 3$ then return(1) end if

Finally, we can solve the overflow-detection problem associated with addition efficiently by use of the sign functions proposed in Sects. 4 and 5 (See Appendix D).

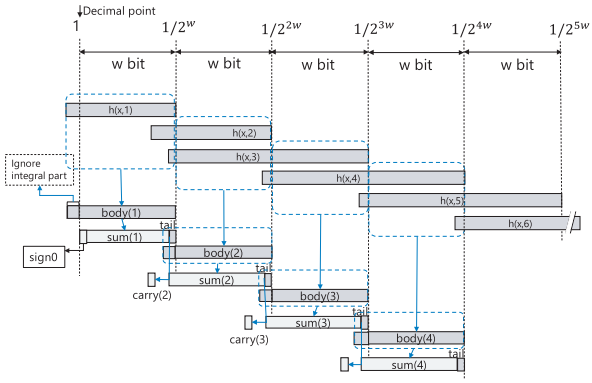


Fig. 5 Bit alignment of steps 7-20 of SDPS for $n = 3$.

6. Evaluation

6.1 Computational Complexity

6.1.1 Probability Derived from d-th Approximation

First, we derive the probability that a sign is determined from $G(x, d)$. In this case, a residual error is bounded by $e(d)$ from (4), while in SDPS, the error is bounded by a word boundary. Therefore, the probability derived here is not equal to that of SDPS, but the derivation process here will be applied to derive a probability of SDPS in the next Sect. 6.1.2. We assume that the distribution of input x is uniform.

From Fig. 1, the probability that sign of x is determined is estimated as

$$\varphi_d = 1 - 2e(d)$$

under the condition that the error is $e(d)$.

Let p_d denote the probability that a sign is determined when the d-th term is computed but not before this term. Then, p_d is represented by

$$p_d = \varphi_d - \varphi_{d-1},$$

where $d = 0, 1, 2, \cdots$ and $\varphi_{-1} = 0$. The upper bound of (4) leads to

$$p_0 = 1 - 2\sum_{i=1}^{n}\left(1 - \frac{1}{m_i}\right)\left(\frac{\mu_i}{2^w}\right)$$

$$p_d = 2\sum_{i=1}^{n}\left(1 - \frac{1}{m_i}\right)\left(\frac{\mu_i}{2^w}\right)^d\left(1 - \frac{\mu_i}{2^w}\right) \quad (d \geq 1).$$

p_d decreases exponentially as d increases. The expected number of operations can be computed using this probability.

6.1.2 Probability Derived from Error at Word Boundary

Next, we derive probabilities representing SDPS and SDRT. To represent both probabilities with a single formula, let us

Table 1 Number of operations in SDPS (left) and SDRT (right).

Loop count j	Step	Mul	Add
0	0	n	–
	1	–	$n-1$
$1,\cdots,n$	6	n	$n-1$
	8	–	1
Expectation 1		$2n$	$2n-1$
Expectation 2		$\left(1+\frac{1}{V}\right)n$	$\left(1+\frac{1}{V}\right)n-1$

Loop count j	Step	Mul	Add
0	0	n	–
	1	–	$n-1$
	2	n	$n-1$
$1,\cdots,n+1$	6	n	$n-1$
	8, 16	–	2
Expectation 1		$3n$	$3n-1$
Expectation 2		$\left(1+\frac{2}{V}\right)n$	$\left(1+\frac{2}{V}\right)n-1$

replace the loop variable with j. Let p_j denote the probability that a sign is detected at the j-th loop, where $j = 0$ corresponds to the process before the loop. For SDPS, j is the same as the loop variable k used in Fig. 2, whereas for SDRT, j and k have the relationship $j = k - 2$, from Fig. 4. In SDRT, the maximum value of j is $n + 1$, which is one larger than that of SDPS. In our analysis, we assume that $p_{n+1} = 0$. As a result, the same formula can be used for SDPS and SDRT. The assumption above is justified by noting that $p_{n+1} < p_n$ and p_n is negligibly small if w is large enough.

Now, we derive the probability according to the process presented in the previous Sect. 6.1.1. Let e_j be the upper bound of the error in the j-th loop. Then

$$e_j = \frac{a}{N^j},$$

where $N = 2^w$, $a = 1$ for SDPS, and $a = 2$ for SDRT. Thus,

$$\varphi_j = 1 - 2e_j = 1 - \frac{2a}{N^j}.$$

With $\varphi_0 = 0$, for $1 \le j < n$, we obtain

$$p_j = \varphi_j - \varphi_{j-1}.$$

If we assume that a sign is detected for all input up to $j = n$, the probability is described as follows:

$$\begin{aligned} p_1 &= 1 - \frac{2a}{N} \\ p_j &= \frac{2a}{N^{j-1}}\left(1 - \frac{1}{N}\right) \quad (\text{for } 2 \le j \le n-1) \\ p_n &= \frac{2a}{N^{n-1}} \end{aligned}$$

6.1.3 Expected Number of Steps of Computation

Table 1 summarizes the number of multiplications (Mul) and additions (Add) executed at each step in the j-th loop of SDPS (left) and SDRT (right). Each table has an entry, Expectation 1, which presents the expected number of operations computed with the probability derived in 6.1.2. As an example, the average number of multiplications in SDPS is computed as

$$\begin{aligned} &\sum_{j=1}^{n} p_j \cdot (1+j)n \\ &\quad = n + n\sum_{j=1}^{n-1} p_j \cdot j \\ &\quad = n + n\left\{1 - 2a + 2a\left(\frac{1-\left(\frac{1}{N}\right)^n}{1-\frac{1}{N}}\right)\right\} \\ &\quad \approx 2n. \end{aligned}$$

This result implies that the algorithm halts mostly at the first loop if N is sufficiently large. In other words, the probability that the algorithm is continued after the first loop is negligibly small. The worst-case complexity is no more than $O(n^2)$, which occurs when the sign is not determined until the last loop.

The standard deviation for the number of multiplications is derived as

$$\sigma \approx \sqrt{\frac{6a}{N}} \cdot n.$$

This represents a very steep distribution around the average.

The result in Table 1 leads to the following Theorem.

Theorem 4:

If the input to SDPS and SDRT is chosen uniformly and at random, and if $N = 2^w$ is sufficiently larger than 1, then the expected number of operations is $O(n)$ multiplications and $O(n)$ additions ■

Furthermore, with n multipliers operating in parallel, the expected number of multiplications becomes $O(1)$. Similarly, with an n-input adder, the order of additions turns into $O(1)$. In addition, multiplication at step 0 becomes unnecessary if x can be represented by ξ_i instead of $\{x\}_B$.

A numerical experiment was done to confirm the validity of the theoretical expression of the probability. Figure 6

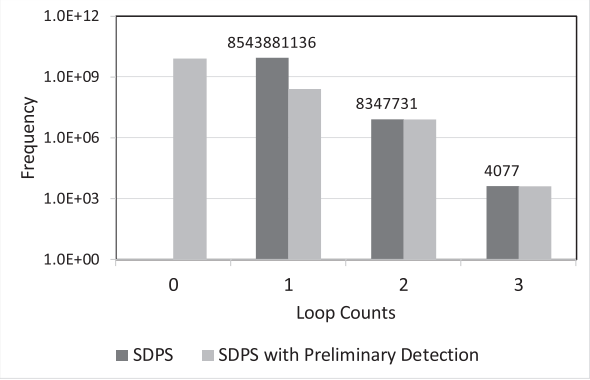


Fig. 6 Frequency of loop 2.

is a histogram that shows the frequency of the loop number at which the algorithm SDPS halts when the input x varies from 0 to $M-1$. The dark gray graph is SDPS of interest here and the light gray graph will be discussed in 6.1.4. The vertical axis is shown on a log scale.

We select relatively small parameters, $n = 3$, $w = 11$, and $(\mu_1, \mu_2, \mu_3) = (1, 3, 5)$, so that an exhaustive experiment on x is possible. The result agrees well with the expectation computed from the probability. In fact, the error relative to the theoretical result is less than 0.4%. The frequency of a loop count of 1 is much larger than the sum of other frequencies. The ratio is about $(1 - 2a/N) : (2a/N) \approx 1000 : 1$ both experimentally and theoretically, for our parameters. It is only near $x = 0, M/2, M$ that a loop count more than 2 is seen. A similar result is obtained for SDRT.

6.1.4 Preliminary Detection for Improvement

To reduce the computation steps further, we propose to add preliminary detection steps that evaluate the first several bits of the first approximate term immediately after it is computed. Here is the pseudocode of the preliminary detection.

s1: $low \leftarrow \langle F \rangle_{2^w}$
s2: $sign \leftarrow low \gg (w-1)$
s3: $tmp \leftarrow low \bigwedge Mask$
s4: if $tmp \neq Mask$ then return($sign$) end if

To improve SDPS, steps 2 and 3 in Fig. 2 should be replaced by the above code. In this case, symbol F at step s1 should be replaced by gx_k. As for SDRT, the above code should be added immediately before step 2. This time, F should be replaced by $h2$. The constant $Mask$ is defined as follows:

$$v = \left\lceil \log_2 \left(\sum_{i=1}^{n} \mu_i \right) \right\rceil,$$
$$Mask = 2^{w-1} - 2^v.$$

$Mask$ is designed so that the MSB and v least significant bits are zero and the other bits are one. In the preliminary detection steps, tmp has a bit string that is cut out by $Mask$ from low (at step s3). If the bit string is not all 1s, the sign bit is fixed and the algorithm halts. Otherwise, if the bit string is all 1s, the algorithm proceeds to compute the next term.

Let p'_j denote the probability that the algorithm with preliminary detection halts in the j-th loop. Then,

$$p'_0 = 1 - \frac{1}{V},$$

where

$$V = 2^{w-v-1}.$$

The rest are

$$p'_1 = \frac{1}{V} - \frac{2a}{N},$$
$$p'_j = p_j \ (\text{for } \ 2 \le j \le n).$$

p_j is the probability defined in 6.1.2. The expectation computed using p'_j is shown in Expectation 2 of Table 1. The experimental result for SDPS with preliminary detection is shown in the light gray graph in Fig. 6.

6.1.5 Optimality

We next discuss the optimality of our algorithm in terms of computational complexity. According to Theorem S ([7], p.291), we must use all elements of $\{x\}_B$ to compute a correct sign. Suppose we use an arbitrary binary operation that has two inputs and one output, a typical example of which is multiplication or addition. Since $\{x\}_B$ has n elements, at least $(n-1)$ operations are necessary to obtain an output that depends on all the elements. Therefore, the number of operations for sign detection cannot be less than $(n-1)$. In Table 1, Expectation 2 for multiplication is slightly greater than n. This shows that our algorithms are very nearly optimal.

6.2 Size of Lookup Table

We evaluate the memory size for a lookup table including a constant $\{W\}_B$. Let S_G and S_H denote the memory sizes for SDPS and SDRT, respectively. Then,

$$\begin{aligned} S_G &= n(n+1)w \approx (n+1)\log_2 M, \\ S_H &= n(n+3)w \approx (n+3)\log_2 M \\ &= \frac{(\log_2 M)^2}{w} + 3\log_2 M \quad \text{(bit)} \\ &\approx O\left((\log_2 M)^2\right). \end{aligned}$$

We assume here that each lookup table is a sequence of words. S_H is larger than S_G, shown in Fig. 7, with parameters $w = 32$, 64, 96, and 128. Even in the most memory-consuming case, $w = 32$, our sign detection can be implemented with 4.38 KB and 98.1 KB memories for $\log_2 M = 1000$ and 5000 bits, respectively. On the other hand, the respective memory size required by the method in [9] are more than 200 KB and 2 MB (Fig. 6(b) of [9]). Hence, our algorithm reduces the memory size by a factor of at least 1/20 for $\log_2 M = 5000$ bits. Graphs of memory

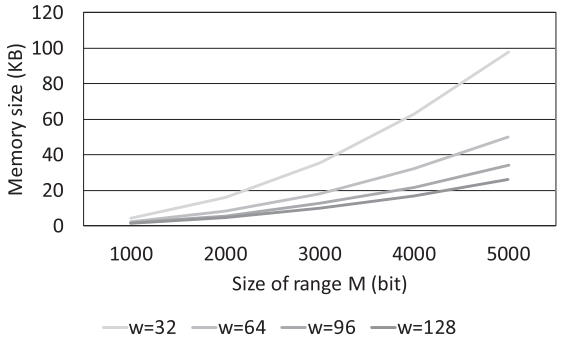


Fig. 7 Memory size of lookup table for SDRT.

size for [9] are out of the range of Fig. 7. In [9], memory size is evaluated as

$$O\left((\log_2 M)^3 / (\log_2 \log_2 M)^2\right).$$

Since the denominator increases very slowly, the degree of this equation in $\log_2 M$ can be approximated by 3, while memory size of our algorithms is degree 2.

Let S_V denote the memory size of Vu's method [8]. Then, it follows that

$$\begin{aligned} S_V &\approx n2^w \log_2 nM \\ &\approx 2^w \left\{ \frac{(\log_2 M)^2}{w} + \left(\frac{\log_2 n}{w}\right) \log_2 M \right\} \\ &\approx O\left(\sqrt[n]{M} \cdot (\log_2 M)^2\right). \end{aligned}$$

Compared with S_H, S_V is about 2^w times the size. Vu's algorithm stores all entries corresponding to all values of x_i, while our algorithm stores only one representative value and produces the variation by multiplying the value ξ_i that depends on the value of x_i. This results in a significant memory reduction.

7. Conclusion

We have proposed efficient sign-detection algorithms, SDPS and SDRT, that compute via the Chinese remainder theorem using approximate reciprocals. The average computational complexity of the algorithms is $O(n)$, where n is the number of base elements. The size of lookup table is reasonably small and at most $(n+3)\log_2 M$ bits. To the best of our knowledge, the proposed algorithms realize the best efficiency and the least memory. At least, they are superior to all algorithms that were evaluated in [9]. We conjecture that there is little hope for finding a substantially better method with an order $< O(n)$, for the same reason as mentioned by Knuth. The proposed algorithms make it possible to efficiently compare two integers in RNS. Now we can implement, in RNS, procedures that are important but have been circumvented due to inefficiency of comparison. Such procedures include the binary extended Euclidean algorithm and the final subtraction of Montgomery multiplication. The validity of the proposed algorithms is confirmed by computer experiment. Further study is necessary to evaluate the proposed algorithms on FPGA or ASIC architectures.

Acknowledgments

Part of this paper is based on results obtained from a project, JPN16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] L. Sousa, S. Antao, and P. Martins, "Combining residue arithmetic to design efficient cryptographic circuits and systems," IEEE Circuit Syst. Mag., vol.16, no.4, pp.6–32, 2016.
[2] J.-C. Bajard, J. Eynard, and N. Merkiche, "Montgomery reduction within the context of residue number system arithmetic," J. Cryptogr. Eng., vol.8, no.3, pp.189–200, Springer, 2018.
[3] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel Montgomery multiplication," EUROCRYPT2000, LNCS1807, pp.523–538, Springer, 2000.
[4] K. Bigou and A. Tisserand, "Improving modular inversion in RNS using the plus-minus methods," CHES2013, LNCS8086, pp.223–249, Springer, 2013.
[5] K. Bigou and A. Tisserand, "Binary-ternary plus-minus modular inversion in RNS," IEEE Trans. Comput., vol.65, no.11, pp.3495–3501, Nov. 2016.
[6] H.L. Garner, "The residue number system," IRE Trans. Electron. Comput., vol.EC-8, no.2, pp.140–147, 1959.
[7] D.E. Knuth, The Art of Computer Programing, vol.2, 3rd ed., Addison-Wesley, 1997.
[8] T.V. Vu, "Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding," IEEE Trans. Comput., vol.C-34, no.7, pp.646–651, July 1985.
[9] D.S. Phatak and S.D. Houston, "New distributed algorithms for fast sign detection in residue number systems (RNS)," J. Parallel Distrib. Comput., vol.97, pp.78–95, 2016.
[10] K. Nave, A.S. Molahosseini, and M. Esmaeildoust, "How to teach residue number system to computer scientists and engineers," IEEE Trans. Educ., vol.54, no.1, pp.156–163, Feb. 2011.
[11] S. Kumar and C.-H, Chang, "A new fast and area-efficient adder-based sign detector for RNS $\{2^n-1, 2^n, 2^n+1\}$," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.24, no.7, pp.2608–2612, 2016.
[12] L. Sousa and P. Martins, "Sign detection and number comparison on RNS 3-moduli sets $\{2^n-1, 2^{n+x}, 2^n+1\}$," Circuits Syst. Signal Process., vol.36, no.3, pp.1224–1246, 2017.
[13] A. Hiasat, "A reverse converter and sign detectors for an extended RNS five-moduli set," IEEE Trans. Circuits Syst. I, Reg. Papers, vol.64, no.1, pp.111–121, 2017.
[14] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," CHES2010, LNCS6225, pp.48–64, Springer, 2010.
[15] G.X. Yao, J. Fan, R.C.C. Cheung, and I. Verbauwhede, "Novel RNS parameter selection for fast modular multiplication," IEEE Trans. Comput., vol.63, no.8, pp.2099–2105, Aug. 2014.
[16] S. Kawamura, Y. Komano, H. Shimizu, and T. Yonemura, "RNS Montgomery reduction algorithms using quadratic residuosity," J. Cryptogr. Eng., vol.9, pp.313–331, Springer, 2019.
[17] G.C. Cardarilli, M. Re, R. Lajacono, and G. Ferri, "A systolic architecture for high-performance scaled residue to binary conversion," IEEE Trans. Circuits Syst. I, Fundam. Theory Appl., vol.47, no.10, pp.1523–1526, 2000.

Appendix A: Proof of Theorem 1

At first, if $x = 0$, it follows that $G(0,d) = G(0) = 0$ from

condition (iii). Theorem 1 holds in this case.

If $x \neq 0$, we will show that $\lfloor G(x) \rfloor = \lfloor G(x, \delta) \rfloor$ holds for $\forall x \in [1, M-1]$. From $G(x, d) = G(x) - e(x, d)$ and $0 \leq e(x, d) \leq e(d)$, we can derive

$$G(x) - e(d) \leq G(x, d) \leq G(x).$$

If we substitute the relation

$$\begin{aligned} G(x) &= \langle G(x) \rangle_1 + \lfloor G(x) \rfloor \\ &= \frac{x}{M} + \lfloor G(x) \rfloor, \end{aligned}$$

to this equation, we get the following equation.

$$\frac{x}{M} - e(d) \leq G(x, d) - \lfloor G(x) \rfloor \leq \frac{x}{M} \tag{A·1}$$

From now on, we assume $d = \delta$, which satisfies the assumption of Theorem 1. Under this condition, the lower bound of (A·1) is assessed as follows:

$$\frac{x}{M} - e(\delta) \geq \frac{x}{M} - \frac{1}{2M} = \frac{2x-1}{2M} \geq \frac{1}{2M} > 0. \tag{A·2}$$

We apply condition $x \geq 1$, which comes from our temporal assumption, $x \neq 0$. The upper bound of (A·1) can be evaluated as

$$\frac{x}{M} \leq \frac{M-1}{M} < 1. \tag{A·3}$$

From (A·1)–(A·3), it follows that

$$0 < G(x, \delta) - \lfloor G(x) \rfloor < 1.$$

This means that

$$\lfloor G(x) \rfloor = \lfloor G(x, \delta) \rfloor.$$

This relation justifies the equation

$$\begin{aligned} G(x, \delta) - \lfloor G(x) \rfloor &= G(x, \delta) - \lfloor G(x, \delta) \rfloor \\ &= \langle G(x, \delta) \rangle_1. \end{aligned}$$

Substituting this relation and $d = \delta$ to (A·1), we obtain

$$\frac{x}{M} - e(\delta) \leq \langle G(x, \delta) \rangle_1 \leq \frac{x}{M}. \tag{A·4}$$

Now, we evaluate the range of $\langle G(a, \delta) \rangle_1$ using (A·4) when a is in the range of a positive number as

$$a \in \{x | 1 \leq x < M/2, x \text{ is an integer}\}.$$

The lower bound on (A·4) is derived by (A·2), and the upper bound is

$$(\text{upper bound}) = \frac{x}{M} < \frac{(M/2)}{M} = \frac{1}{2}.$$

Thus, we obtain

$$0 < \langle G(a, \delta) \rangle_1 < \frac{1}{2}.$$

Therefore, $b_{-1} = 0$ holds for $\langle G(a, \delta) \rangle_1$.

Similarly, if

$$b \in \{x | M/2 < x \leq M-1, x \text{ is an integer}\},$$

then

$$\min(b) = \begin{cases} M/2 + 1, & \text{if } M \text{ is even,} \\ M/2 + 1/2, & \text{if } M \text{ is odd.} \end{cases}$$

The upper bound of (A·4) is assessed by (A·3) and the lower bound of (A·4) is

$$(\text{lower bound}) = \frac{x}{M} - e(\delta) \geq \frac{\min(b)}{M} - \frac{1}{2M} \geq \frac{1}{2}.$$

Thus,

$$\frac{1}{2} \leq \langle G(b, \delta) \rangle_1 < 1$$

holds. Therefore, $b_{-1} = 1$ holds for $\langle G(b, \delta) \rangle_1$. Theorem 1 is proven when $x \neq 0$ as well.

(Q.E.D.)

From the above discussion, when M is even, we can replace the assumption of Theorem 1, $e(\delta) < 1/2M$, by $e(\delta) < 1/M$.

Appendix B: Upper Bound of *high*(*k*)

Lemma 2 ensures that $high(k) < 2^w$ holds for $1 \leq k \leq n$.

Lemma 2:

If

$$e(n) \leq \frac{1}{2M},$$

then for $1 \leq k \leq n$,

$$high(k) \triangleq \left\{ \left(\sum_{i=1}^{n} \xi_i \mu_i^k \right) \gg w \right\} < 2^w.$$

■

Proof:

The conclusion of Lemma 2 is equivalent to

$$\sum_{i=1}^{n} \xi_i \mu_i^n < 2^{2w} \tag{A·5}$$

holding when k has a maximum value of n. Equation (A·5) is to be proven.

From $m_i \leq 2^w$, we obtain

$$\sum_{i=1}^{n} \frac{\xi_i}{2^w} \left(\frac{\mu_i}{2^w} \right)^{n+1} \leq \sum_{i=1}^{n} \frac{\xi_i}{m_i} \left(\frac{\mu_i}{2^w} \right)^{n+1}.$$

Since the right-hand side equals $e(x, n)$, we obtain

$$e(x, n) \leq e(n) \leq \frac{1}{2M}$$

from the assumption of Lemma 2. Thus,

$$\sum_{i=1}^{n} \frac{\xi_i}{2^w} \left(\frac{\mu_i}{2^w}\right)^{n+1} \le \frac{1}{2M}$$

holds. This equation can be modified as

$$\begin{aligned}\sum_{i=1}^{n} \xi_i \mu_i^{n+1} &\le \frac{2^{(n+2)w}}{2M} \\ &= \frac{2^{2w}}{2\prod_{i=1}^{n}\left(1-\frac{\mu_i}{2^w}\right)} \\ &\approx 2^{2w-1} \\ &< 2^{2w}. \end{aligned} \tag{A·6}$$

If μ_i is a non-negative integer and n is positive integer,

$$\mu_i^n \le \mu_i^{n+1}$$

holds. This proves (A·5).

(Q.E.D.)

Lemma 3 ensures that $high(1) < 2^{w-1}$.

Lemma 3:
If

$$e(n) \le \frac{1}{2M},$$

then, for $k = 1$, it holds that

$$high(1) \triangleq \left\{\left(\sum_{i=1}^{n} \xi_i \mu_i\right) \gg w\right\} < 2^{w-1}.$$

■

Proof:
It suffices to show that

$$\sum_{i=1}^{n} \xi_i \mu_i < 2^{2w-1}. \tag{A·7}$$

We will assess the left-hand side by case. Suppose $n \ge 2$ and $\mu_i \ne 1$; then $2\mu_i \le \mu_i^2$ holds for any i, even if $\mu_i = 0$ for some i. With these in mind, we have the following cases.
Case 1: $\mu_i \ne 1$.

$$2^n \sum_{i=1}^{n} \xi_i \mu_i \le \sum_{i=1}^{n} \xi_i \mu_i^{n+1}$$

From (A·6) of the proof of Lemma 2, the upper bound is modified as

$$\begin{aligned}2^n \sum_{i=1}^{n} \xi_i \mu_i &< 2^{2w} \\ \sum_{i=1}^{n} \xi_i \mu_i &< 2^{2w-n}\end{aligned}$$

Note that $n \ge 2$, so this equation proves (A·7).
Case 2: when $\mu_1 = 1$.

$$2^n \sum_{i=2}^{n} \xi_i \mu_i + \xi_1 \le \sum_{i=1}^{n} \xi_i \mu_i^{n+1}$$

From (A·6) of the proof of Lemma 2, the following equation is derived.

$$\sum_{i=2}^{n} \xi_i \mu_i < \frac{1}{2^n}\left(2^{2w} - \xi_1\right) \le 2^{2w-n}$$

This proves (A·7) and thus Lemma 3.

(Q.E.D.)

Appendix C: First and Second Entries of Reciprocal Table

The elements of a reciprocal table, $h_i(k)$, can be computed by the following recurrence formula, sequentially from $k = 1$.

$$h_i(k) = \left\lfloor \left(\frac{1}{m_i} - \sum_{j=1}^{k-1} h_i(j) 2^{-jw}\right) 2^{wk} \right\rfloor$$

Substituting a power series for $1/m_i$ leads to

$$h_i(k) = \left\lfloor \left(\frac{1}{2^w} \sum_{l=0}^{\infty} \left(\frac{\mu_i}{2^w}\right)^l - \sum_{j=1}^{k-1} h_i(j) 2^{-jw}\right) 2^{wk} \right\rfloor.$$

If $k = 1$ and we substitute $\varepsilon_i = \mu_i / 2^w$, then

$$h_i(1) = \left\lfloor \frac{1}{1-\varepsilon_i} \right\rfloor = \left\lfloor 1 + \frac{\varepsilon_i}{1-\varepsilon_i} \right\rfloor = 1.$$

The last equation is due to $\varepsilon_i < 1/2$.

Similarly, if $k = 2$, we can derive

$$h_i(2) = \left\lfloor \mu_i + \frac{\mu_i \varepsilon_i}{1-\varepsilon_i} \right\rfloor = \mu_i + \left\lfloor \frac{1}{2^w} \cdot \frac{\mu_i^2}{1-\varepsilon_i} \right\rfloor.$$

To find the condition in which the value in the last floor symbol is less than 1, we define f as $f(\mu_i) = \mu_i^2 + \mu_i - 2^w$. The positive root of $f(\mu_i) = 0$ is given by

$$s = \frac{\sqrt{2^{w+2}+1} - 1}{2}.$$

We can summarize the result as

$$h_i(2) = \begin{cases} \mu_i, & (0 \le \mu_i < s) \\ \mu_i + 1, & \left(s \le \mu_i < 2^{\lfloor w/2 \rfloor}\right). \end{cases}$$

Since s is rather close to $2^{\lfloor w/2 \rfloor}$, if we choose μ_i at random, $h_i(2) = \mu_i$ holds with a very high probability.

Appendix D: Overflow Detection of Modular Addition

An overflow (OF) detection of modular addition is known as a relevant problem for sign detection. To realize an efficient OF detection, we represent an integer x by $\{x\}_B$ appended with s_x, a sign bit of Definition 1. As shown in Fig. A·1, the OF flag can be computed efficiently with a single call to the sign function.

Add($\{x\}_B$, s_x, $\{y\}_B$, s_y,B)
Required OF=1 if $x + y \geq M$. OF=0, otherwise.
1: $\{z\}_B \leftarrow \{x\}_B + \{x\}_B$
2: $s_z \leftarrow$ Sign($\{z\}_B$,B)
3: OF $\leftarrow$ NOT(s_x XOR s_y) AND s_x+(s_x XOR s_y) AND (NOT(s_z))
4: return($\{z\}_B$, s_z, OF)

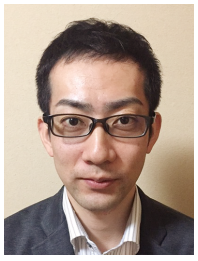
Fig. A· 1 Overflow detection of addition.

Shinichi Kawamura received B.E., M.E., and D.E. degrees in Electronic Engineering from the University of Tokyo in 1983, 1985, and 1996, respectively. He joined Toshiba Corporation in 1985, and retired from the company in 2020 as a Senior Fellow. Currently, he is a Deputy Director of the Cyber Physical Security Research Center (CPSEC) at the National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan. His research interests are in cryptography and its application to security systems. Dr. Kawamura is a Fellow of IEICE, a Senior Member of IEEE, a Senior Member of IPSJ, and a member of IACR. He was a recipient of IEICE's Young Researchers' Award in 1993, the Distinguished Service Award from IEICE Engineering Sciences Society in 2006, and an IPSJ Specially Selected Paper Certificate in 2014.

Yuichi Komano was born in 1978. He received his M.S. and D.Sci. degrees from Waseda University in 2003 and 2007, respectively. He belongs to the Corporate R&D center of Toshiba corporation since 2003. His research interest includes the cryptography and information security. He is a senior member of the IEICE and IPSJ, and a member of the IACR, IEEE and ACM.

Hideo Shimizu was born in 1964. He received his M.E. and D.E. degrees from Kanazawa Institute of Technology, Ishikawa, Japan, in 1990 and 1994, respectively. He joined Toshiba Corporation in 1994. From 1999 to 2000, he was a researcher at the Information & Communication Security Project of Telecommunications Advanced Organization of Japan. He has been engaged in cryptography and information security.

Saki Osuka received the M.E. degrees from Nara Institute of Science and Technology, Nara, Japan in 2019. She is currently working toward the Ph.D. degree in information sciences at Nara Institute of Science and Technology, Nara, Japan. Her research interests include electromagnetic compatibility and information security. Ms. Osuka is a member of IEEE.



Daisuke Fujimoto received B.E., M.E., and Ph.D. degree from Kobe University, Japan, in 2009, 2011 and 2014, respectively. He is currently an assistant professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. He is also a visiting assistant professor in the Institute of Advanced Sciences, Yokohama National University. His research interests include hardware security and implementation of security cores. He is a member of IEEE and IEICE.

Yuichi Hayashi received Ph.D. degree in information sciences from Tohoku University, Sendai, Japan, in 2009. He is currently a Professor of Nara Institute of Science and Technology. His research interests include electromagnetic compatibility and information security. Prof. Hayashi is the Chair of EM Information Leakage Subcommittee in IEEE EMC Technical Committee 5.

Kentaro Imafuku received his Ph.D. degree from Waseda University. After professional experiences in Waseda university, University of Rome Tor Vergata (JST/JSPS overseas young research fellowship), and the University of Tokyo, he is working in National Institute of Advanced Industrial Science and Technology. He is conducting theoretical researches on applications of physics and information theory to computer science and engineering.